TEAM D

General testing method:

1. Method 1
    a. Import our test cases into group's project.
    b. Update/remove exception messages, and format expected test results to match group's formatting of their output results.
    c. Run tests - check tests which fail against group's assumptions and requirements provided.
2. Method 2
    a. Run ShellImpl and type commands
    b. Write failing test cases

Bug Report Number	Description	Testcase	Comments (leave empty)
1	Pwd: pwd does not throw any exception for null arguments. Same behaviour observed for sort.	*// pwd with null args* @Test public void runThrowsExceptionNullArguments() { assertThrows(PwdException.class, () -> pwd.run(null, stdin, stdout)); } *//sort with null args* @Test public void runThrowsExceptionNullArguments() { assertThrows(SortException.class, () -> sa.run(null, stdin, stdout), ""); }	

2	Pwd: pwd accepts additional arguments. Command format in requirements is *pwd*. No [optional text] is specified in the requirements (e.g. date [+format] for date allows for optional additional format argument) therefore pwd should not accept any additional arguments. This was not addressed in the assumptions either.	*// pwd hello* @Test public void runThrowsExceptionIncorrectNumArguments() { String[] args = {"hello"}; assertThrows(PwdException.class, () -> pwd.run(args, stdin, stdout)); }	
3	Pwd: pwd with additional whitespace arguments throws ShellException, and doesn't seem to recognize pwd argument at all. This behaviour is also seen in all the other applications e.g. *date [/s], echo [/s], etc.*	*// pwd [/s] [/s] where [/s] are whitespaces* *// Throws ShellException: shell: Invalid syntax.* @Test public void runWithWhitespaceArguments() throws AbstractApplicationException, ShellException { Shell shell = new ShellImpl(); ByteArrayOutputStream out = new ByteArrayOutputStream(); String arg = "pwd "; shell.parseAndEvaluate(arg, out); String expectedResult = Environment.currentDirectory; String actualResult = out.toString(); assertEquals(expectedResult,	

		actualResult); } 	
4	Pwd: pwd with a single character additional argument throws a null pointer exception, while pwd with multiple character additional argument does not throw any such exception. Therefore, this behaviour is inconsistent. This behaviour is also seen in all other applications e.g. *echo a, date +, etc.*	*// pwd a* @Test public void runWithSingleCharacterArgument() throws AbstractApplicationException, ShellException { String arg = "pwd a"; shell.parseAndEvaluate(arg, outstream); String expectedResult = Environment.currentDirectory; String actualResult = outstream.toString(); assertEquals(expectedResult, actualResult); } *// date +* @Test public void runSuccessOnlyPlusArgument() throws AbstractApplicationException, ShellException { String arg = "date +"; shell.parseAndEvaluate(arg, outstream); String expectedResult = System.lineSeparator(); String actualResult = outstream.toString(); assertEquals(expectedResult, actualResult); }	
5	Date:	*// date +"%m-%t"* *// Unix: 03-*	

	date accepts invalid fields. In UNIX shell, such invalid fields are ignored and not printed as string literals. Alternatively, an exception should be thrown for such invalid fields.	*// Result: 03-%t* @Test public void runThrowsExceptionIncorrectTag() { String[] args = {"+%m-%t"}; assertThrows(DateException.class, () -> date.run(args, stdin, stdout)); }	
6	Date: date with 2 % signs should either throw exception for invalid tag, or print only a single % similar to UNIX shell.	*// date +"%%"* *Expected: % / exception* *Result: %%* @Test public void runThrowsExceptionTwoPercentArgument() throws AbstractApplicationException, ShellException { String[] args = {"+%%"}; assertThrows(DateException.class, () -> date.run(args, stdin, stdout)); }	
7	Echo: echo does not throw any exception when stdout is null.	*// stdout == null* @Test public void runThrowsExceptionNullStdout() { String[] args = {"hello"}; assertThrows(EchoException.class, () -> echo.run(args, stdin, null)); }	
8	Echo: echo does not accept empty	*// echo* @Test public void runSuccessNoArgument() throws	

	arguments. Command format in requirements is *echo [ARGS]*, therefore additional arguments for echo are optional. UNIX shell also accepts empty arguments for echo, printing a newline. This has not been addressed in Assumptions either.	EchoException { String expectedResult = ""; String[] args = {}; echo.run(args, stdin, stdout); assertEquals(expectedResult, stdout.toString()); }	
9	Exit: Exit does not terminate execution with correct status == 0.	*// exit* @Test public void runSuccessCorrectExitStatus() throws ExitException { String[] args = {}; try { exit.run(args, stdin, stdout); } catch (ExitSecurityManager.ExitSecurityException e) { int status = e.getStatus(); assertEquals(0, status); } }	
10	Exit: exit accepts additional arguments when user input is typed directly into running execution of ShellImpl, but throws an exception when	*// exit hello* @Test public void runWithAdditionalArguments() throws ExitException { String[] args = {"exit hello"}; try { exit.run(args, stdin, stdout);	

	written as a test. In fact, all exit tests throw an ExitException. Therefore, behaviour is contradictory. Exit should terminate without throwing any exception.	} catch (ExitSecurityManager.ExitSecurityException e) { int status = e.getStatus(); assertEquals(0, status); } }	
11	Wc: Number of bytes computed is incorrect.	*// wc -c test1.txt* *// Expected: 52* *// Result:: 46* @Test public void runSuccessSingleByteTag() throws WcException { String expectedResult = "52 " + TEST_1_PATH + STRING_NEWLINE; String[] args = {"-c", TEST_1_PATH}; wc.run(args, stdin, stdout); assertEquals(expectedResult, stdout.toString()); }	
12	Wc: wc - is recognized as a valid option, but wc -- is recognized as an invalid option. wc -- is valid in the Unix shell.	*// wc --* @Test public void runSuccessTwoDash() throws WcException { String expectedResult = "6 12 52 " + TEST_1_PATH + STRING_NEWLINE; String[] args = {"--", TEST_1_PATH}; wc.run(args, stdin, stdout); assertEquals(expectedResult, stdout.toString()); }	

13	Wc: Exception messages not very clear, for directory, and file does not exist. Simply says “IO not working” for both.	*// wc: Directory* @Test public void runThrowsExceptionDirectory() { String[] args = {"hackathon_tests" + File.separator + "resources" + File.separator + "wcFolder"}; WcException e = assertThrows(WcException.class, () -> wc.run(args, stdin, stdout)); assertTrue(e.getMessage().contains("is a directory")); }	
14	Wc: Cannot terminate stdin (user input). When ctrl-d is tried, program hangs, and needs to be manually terminated. This behaviour is also observed for all other applications using stdin such as *sort, sed, etc.*	*// wc* @Test public void runSuccessStdin() { assertTimeoutPreemptively(ofMillis(1000), () -> { String arg = "wc"; Shell shell = new ShellImpl(); ByteArrayOutputStream outstream = new ByteArrayOutputStream(); InputStream in = System.in; String expectedResult = "2 2 12 "; String str = "hello" + System.lineSeparator() + "world" + System.lineSeparator(); System.setIn(new ByteArrayInputStream(str.getBytes())); shell.parseAndEvaluate(arg, outstream); String actualResult =	

		outstream.toString(); assertEquals(expectedResult, actualResult); }); }	
15	Wc: Incorrect output for different file formats, such as image files. Assumptions do not specify any valid/invalid file formats either.	*// wc smiley.jpg* *// Expected: 595 2004 68351 smiley.jpg* *// Result: 753 893 120599 smiley.jpg* @Test public void runSuccessDifferentFileFormat() throws WcException { String expectedResult = "595 2004 68351 " + TEST_2_PATH; String[] args = {TEST_2_PATH}; wc.run(args, stdin, stdout); assertEquals(expectedResult, stdout.toString()); }	
16	Wc: When 1 valid and 1 invalid file is provided, exception is thrown and the valid file is not evaluated. This is not addressed in the Assumptions, and does not follow standard Unix shell behaviour. As mentioned in (16), exception message is also not clear, just says “IO not working”.	*// wc test1.txt test100.txt where test1.txt is valid and test100.txt is invalid.* @Test public void runSuccessValidWithInvalidFile() throws WcException { String expectedResult = "52 " + TEST_1_PATH; String[] args = {"-c", TEST_1_PATH, "test1000.txt"}; wc.run(args, stdin, stdout); assertEquals(expectedResult, stdout.toString()); }	

17	IORedirection: Input and output redirection not reading provided file path. The first character is removed from filePath Same behaviour observed for wc.	Team8ShellImplTest @Test public void inputRedirectionTest() throws AbstractApplicationException, ShellException { String expected = ""; shell.parseAndEvaluate("cat < findFolder1" + SEPARATOR + "findFile1.txt",stdout); assertEquals(expected, stdout.toString()); } @Test public void outputRedirectionTest() throws AbstractApplicationException, ShellException { shell.parseAndEvaluate("cat findFolder1" + SEPARATOR + "findFile1.txt > output.txt",stdout); assertTrue(Files.exists(IOUtils.resolveFilePath("output.txt"))); } *// wc < test1.txt* where *test1.txt exists* @Test public void runSuccessStdinFile() { assertTimeoutPreemptively(ofMillis(1000), () -> { String arg = "wc < " + TEST_1_PATH; Shell shell = new ShellImpl(); ByteArrayOutputStream outstream = new ByteArrayOutputStream(); InputStream in = System.in;	

		shell.parseAndEvaluate(arg, outstream); String expectedResult = "6 12 52 "; String actualResult = outstream.toString(); assertEquals(expectedResult, actualResult); }); }	
18	Ls: Empty folder name is not recognise when tags are present	Team8ShellImplTest @Test public void LsTestThrowsErrorEmptyFolderNameWithdTag() throws AbstractApplicationException, ShellException { assertThrows(Exception.class, ()->shell.parseAndEvaluate("ls -d", stdout)); } @Test public void LsTestThrowsErrorEmptyFolderNameWithRTag() throws AbstractApplicationException, ShellException { assertThrows(Exception.class, ()->shell.parseAndEvaluate("ls -R", stdout)); } @Test public void LsTestThrowsErrorEmptyFolderNameWithBothT	

		ag() throws AbstractApplicationException, ShellException { assertThrows(Exception.class, ()->shell.parseAndEvaluate("ls -d -R", stdout)); }	
19	Not reading path from Environment for ls, Find and Mkdir	Team8ShellImplTest @Test public void LsTestSuccess() throws AbstractApplicationException, ShellException { String expected = "findFile1.java findFile1.java"; shell.parseAndEvaluate("ls findFolder1",stdout); assertEquals(expected, stdout.toString()); } @Test public void FindTestSuccess() throws AbstractApplicationException, ShellException { String expected = PATH + SEPARATOR + "findFolder1" + SEPARATOR + "findFile1.txt" + STRING_NEWLINE; shell.parseAndEvaluate("find findFolder1 -name 'findFile1.txt'",stdout); assertEquals(expected, stdout.toString()); } @Test public void mkdirTestSuccess() throws AbstractApplicationException, ShellException { shell.parseAndEvaluate("mkdir new",stdout);	

		assertTrue(Files.exists(IOUtils.resolveFilePath("new"))); } 	
20	Ls: “ls” is not detected as empty but “ls ” is.	Team8ShellImplTest @Test public void LsTestThrowsErrorEmptyFolder() throws AbstractApplicationException, ShellException { assertThrows(Exception.class, ()->shell.parseAndEvaluate("ls", stdout)); }	
21	Globbing: Not detected correctly in find and ls	Team8ShellImplTest @Test public void LsTestSuccessGlobbing() throws AbstractApplicationException, ShellException { String expected = PATH + SEPARATOR + "findFolder1:" + STRING_NEWLINE + "findFile1.java findFile1.java" + STRING_NEWLINE + PATH + SEPARATOR + "findFolder2:" + STRING_NEWLINE + "findFile2.java findFile2.txt findFolder2-1"; shell.parseAndEvaluate("ls " + PATH + SEPARATOR + "*",stdout); assertEquals(expected, stdout.toString()); } @Test public void FindTestSuccessGlobbing() throws AbstractApplicationException, ShellException {	

		String expected = PATH + SEPARATOR + "findFolder1" + SEPARATOR + "findFile1.txt" + STRING_NEWLINE; shell.parseAndEvaluate("find " + PATH + SEPARATOR + "* -name 'findFile1.txt'",stdout); assertEquals(expected, stdout.toString()); }	
22	Find: When '*' is present in the file name no file is return. For example 'findFile*' returns empty file when the folder has findFile1.txt file.	Team8ShellImplTest @Test public void FindTestSuccessGlobbingFileName() throws AbstractApplicationException, ShellException { String expected = PATH + SEPARATOR + "findFolder1" + SEPARATOR + "findFile1.java" + STRING_NEWLINE + PATH + SEPARATOR + "findFolder1" + SEPARATOR + "findFile1.txt" + STRING_NEWLINE; shell.parseAndEvaluate("find " + PATH + SEPARATOR + "findFolder1 -name 'findFile1*'",stdout); assertEquals(expected, stdout.toString()); }	
23	Pipe Command: Not working at all	Team8ShellImplTest @Test public void PipeCommandTestSuccess() throws AbstractApplicationException, ShellException { String expected = "2" + STRING_NEWLINE;	

		shell.parseAndEvaluate("echo 'hackathon test' | wc -w",stdout); assertEquals(expected, stdout.toString()); }	
24	Sequence Command: Not working at all	Team8ShellImplTest @Test public void SequenceCommandTestSuccess() throws AbstractApplicationException, ShellException { String oldDir = Environment.currentDirectory; shell.parseAndEvaluate("cd findFolder1; pwd",stdout); assertEquals(oldDir + SEPARATOR + "findFolder1", Environment.currentDirectory); }	
25	Sed: When there are 3 arguments, Shell takes inputs from stdin	sed “s/a/b/” sed.txt empty.txt	
26	Sed: REPLACEMENT argument fails when there are leading spaces, while it is valid for actual Shell implementation.	sed “ s/a/b/” sed.txt	
27	Sed: REPLACEMENT argument	sed “s/a/b/ ” sed.txt	

	fails when there are trailing spaces, while it is valid for actual Shell implementation.		
28	Sed: REPLACEMENT still valid when there is not enough delimiter, while it is invalid for actual Shell implementation.	sed “s/a/b” sed.txt	
29	Sed: REPLACEMENT still valid when there is extra delimiter, while it is invalid for actual Shell implementation.	sed “s/a/b//” sed.txt	
30	Sed: REPLACEMENT invalid when the replacement string is missing and there is no X. While it is valid in actual Shell.	sed “s/a//” sed.txt	
31	Sed: REPLACEMENT invalid when there are leading spaces in front of X. This is valid in actual Shell.	sed "s/a/b/ 2" sed.txt	
32	Sed: When delimiter is a number,	sed "s9a9b99" sed.txt (9 is used as delimiter, and it appears in X. The 9 in X should be treated as X, instead of delimiter.)	

	and it appears in X, it is still treated as delimiter, instead of X. It is treated as X in actual Shell.		
33	Sort: sort -n sort1.txt throws uncaught exception	sort -n sort1.txt	
34	Sort: When -f used and 2 words are compared equal without considering case, then they should be compared as if they are case-sensitive to break the tie. This is not done for Team D.	@Test public void runSuccessOneFileFFlag() throws AbstractApplicationException { String expectedResult = "&&**" + System.lineSeparator()+ "0" + System.lineSeparator()+ "0a" + System.lineSeparator()+ "1" + System.lineSeparator()+ "10" + System.lineSeparator()+ "10a" + System.lineSeparator()+ "1a" + System.lineSeparator()+ "2" + System.lineSeparator()+ "2a" + System.lineSeparator()+ "AABB" + System.lineSeparator()+ "aabb" + System.lineSeparator()+ "ABCD" + System.lineSeparator()+ "abcd"; String[] args = {"-f", SORT3_PATH}; sa.run(args, stdin, stdout); assertEquals(expectedResult, stdout.toString()); }	

35	Sort: When -n is used, numbers with characters (e.g. 10a, 2a) appended behind are not sorted correctly.	@Test public void runSuccessOneFileNFlag() throws AbstractApplicationException { String expectedResult = "&&**" + System.lineSeparator()+ "0" + System.lineSeparator()+ "0a" + System.lineSeparator()+ "1" + System.lineSeparator()+ "1a" + System.lineSeparator()+ "2" + System.lineSeparator()+ "2a" + System.lineSeparator()+ "10" + System.lineSeparator()+ "10a" + System.lineSeparator()+ "AABB" + System.lineSeparator()+ "ABCD" + System.lineSeparator()+ "aabb" + System.lineSeparator()+ "abcd"; String[] args = {"-n", SORT3_PATH}; sa.run(args, stdin, stdout); assertEquals(expectedResult, stdout.toString()); }	
36	Sort: When only - is used as used as flag, it is invalid. But the Team D Shell continues processing.	@Test public void runFailureInvalidFlagDash() { String[] args = {"-", SORT2_PATH}; assertThrows(SortException.class, () -> sa.run(args, stdin, stdout), ""); }	
37	Mkdir:	Team8ShellImplTest	

	No error message when the folder exists	@Test public void mkdirTestThrowExceptionExistingDir() throws AbstractApplicationException, ShellException { assertThrows(Exception.class, ()->shell.parseAndEvaluate("mkdir hackathon_tests", stdout)); }	
38	Mkdir: No error message when the parent folder does not exists.	Team8ShellImplTest @Test public void mkdirTestThrowExceptionInvalidParentDir() throws AbstractApplicationException, ShellException { assertThrows(Exception.class, ()->shell.parseAndEvaluate("mkdir hackathon_test" + SEPARATOR+"new", stdout)); }	
39	Grep: Uncaught and unhandled String index out of bounds exception when only “-” is used as flag.	@Test public void runInvalidFlagDash() { String[] args = {".*LINE.*", "-c", "-i", "-", file1}; assertThrows(GrepException.class, () -> grep.run(args,input,output), """); }	
40	Grep: When invalid flag "-ia" is used,	@Test public void runInvalidFlagAfterValidOne() { String[] args = {".*LINE.*", "-ia", file1};	

	no exception is thrown and Shell continues to process.	assertThrows(GrepException.class, () -> grep.run(args,input,output), ""); }	